Aspects Système de SQL

- SQL immergé
- SQL et accès concurrents
- SQL et Client/Serveur
- SQL et Sécurité

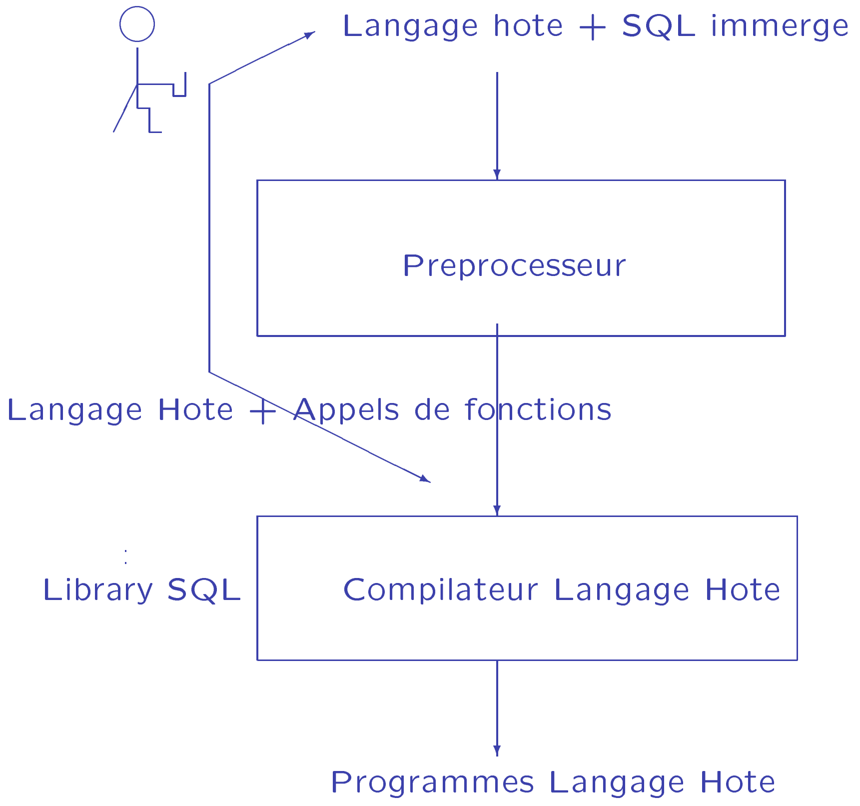
SQL Immergé

- Jusqu'alors : mode ad-hoc

- En pratique rare.

- SQL est utilisé à partir d'un langage hôte.
  - **C**
  - Cobol
  - PL1
  - ...

Architecture Logicielle



Dysfonctionnement (Impedance Mismatch)

Différents systèmes de types

- SQL : noyau modèle relationnel.
  $\Rightarrow$ traite des ensembles de n-uplets.

- Langage hôte (C) autre modèle.

  - pointeurs
  - enregistrements
  - tableaux ...

Interface SQL/Langage Hôte

- Transfert d'information entre la base et le langage au moyen de variables d'échange.

- Pour prévenir le langage hôte d'appels SQL :

```
EXEC SQL <statement>
```

  Le statement SQL à exécuter sera remplacé par une fonction de la library SQL.

```
EXEC SQL INSERT VALUES INTO ...
```

Interface SQL/Langage Hôte

- Une variable spécifique (en SQL2) :

  `SQLSTATE`

  Permet de controler la connection entre les deux mondes.

  - array de 5 char contient '00000' si OK
  - contient '02000' si un n-uplet n'a pu être retrouvé.
  - autres codes voir la doc

La section DECLARE

```
EXEC SQL BEGIN DECLARE SECTION;

...

EXEC SQL END DECLARE SECTION;
```

- Exemple

```
EXEC SQL BEGIN DECLARE SECTION;
     char  nomstudio[15], addrstudio[50];
     char SQLSTATE[6];
     /* En C attention au '\0' ! */
EXEC SQL END DECLARE SECTION;}
```

Utilisation des variables d'échange

- schéma de la relation Studio :

```
Studio(nom, adresse).
```

```
void getStudio() {
1) EXEC SQL BEGIN DECLARE SECTION;
2)      char nomstudio[15], addrstudio[50];
3)      char SQLSTATE[6];
4) EXEC SQL END DECLARE SECTION;}

   /* Demande a l'utilisateur le nom et
      l'adresse du studio et met la reponse
      dans nomstudio et addrstudio */

5) EXEC SQL INSERT INTO Studio(nom, adresse)
6)      VALUES (:nomstudio, :addrstudio);
}
```

Et les requêtes ?

- Ne peuvent pas apparaître telles quelles dans un `EXEC SQL` !

- Pourquoi ? Dysfonctionnement! C ne supporte pas directement les ensembles.

- Que faire ? Deux possibilités :

  1. Si on sait que la requête produit un unique n-uplet comme résultat alors on peut le récupérer par des variables d'échanges (une variable par attribut).
  2. Si la requête produit un ensemble de n-uplet alors il faut utiliser des *curseurs*.

Exemple : premier cas

Schéma :

```
Studio(Nom, Adresse, PresId),

Producteur(PresId, BeneficePres)

void printBenefPres() {
1) EXEC SQL BEGIN DECLARE SECTION;
2)      char Nom-Studio[15];
3)      int  Benef-Pres;
4)      char SQLSTATE[6];
5) EXEC SQL END DECLARE SECTION;

   /* demande d'entrer le nom du studio
      met la reponse dans Nom-Studio */

6) EXEC SQL SELECT BeneficePres
7) INTO :Benef-Pres
8) FROM Studio, Producteur
9) WHERE Studio.Presid = Producteur.Presid
         AND Studio.Nom = :Nom-Studio;
10) if !(strcmp(SQLSTATE, '00000'))
11)     printf(``%s'', Nom-Studio);
12) else ...
}
```

Curseurs

Permettent de lier langage hôte et ensemble de n-uplets.

Pour créer et utiliser un curseur il faut :

- Le déclarer.

```
EXEC SQL DECLARE <curseur> CURSOR FOR
              <requete>
```

- L'ouvrir

```
EXEC SQL OPEN <curseur>
```

- Pouvoir l'exécuter

Curseurs

- Pour exécuter un curseur on il faut opérer un ou plusieurs commandes `FETCH`.

- Une telle commande a pour effet de récupérer (dans des variables d'échange) le n-uplet suivant dans la relation résultat de la requête argument du curseur.

```
EXEC SQL FETCH FROM <curseur> INTO
        <liste de variables>
```

- Fermer le curseur.

```
EXEC SQL CLOSE <curseur>
```

Curseurs : exemple

```
void BenefIntervalles() {
     int i, digits, comptes[15];
EXEC SQL BEGIN DECLARE SECTION;
     int benef;
     char SQLSATE[6];
EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE execCurseur CURSOR FOR
     SELECT BeneficePres FROM Producteur;

EXEC SQL OPEN  execCurseur;
for (i=0; i<15; i++) comptes[i] = 0;
while(1) {
     EXEC SQL FETCH FROM execCurseur INTO  :benef;
     if(!(strcmp(SQLSTE,’’02000’’)) break;
     digits = 1;
     while((benef /=10) > 0) digits++;
     if (digit <= 14) comptes[digits]++;
 }
EXEC SQL CLOSE execCurseur;
for (i=0; i<15; i++)
     printf(‘‘chiffres = %d : nombre de prod = %d\n’’,
                  i, comptes[i]);
}
```

Modifications par curseurs

```
1) void modifBenef() {

2) EXEC SQL BEGIN DECLARE SECTION;
3)      int benef;
4)      char SQLSATE[6];
5) EXEC SQL END DECLARE SECTION;

6) EXEC SQL DECLARE execCurseur CURSOR FOR
7)      SELECT BeneficePres FROM Producteur;

8) EXEC SQL OPEN  execCurseur;

9) while(1) {

10)     EXEC SQL FETCH FROM execCurseur INTO  :benef;

11)     if(!(strcmp(SQLSTE,’’02000’’)) break;
12)     if (benef < 1000)

13)        EXEC SQL DELETE FROM Producteur
                    WHERE CURRENT OF execCurseur;

14)     else

15)         EXEC SQL UPDATE Producteur
                     SET BeneficePres = 2 * BeneficePres
                     WHERE CURRENT OF execCurseur;
    }
16) EXEC SQL CLOSE execCurseur;
}
```

Exemple : ordonner les n-uplets

```
EXEC SQL DECLARE filmStarcurseur CURSOR FOR
    SELECT  titre, annee, studio, nomStar
    FROM  Film, Stars
    WHERE titre = filmTitre AND annee = filmAnnee
    ORDER BY aneee, titre;
```

Protection des accès concurrents

Deux processus distincts : P_1 et P_2

- P_1 lit des n-uplets au moyen du curseur `filmStarcurseur`
- P_2 modifie les relations Film et Stars.

s'exécutent de manière concurrente.
Ceci peut causer un comportement indésirable :

- P_1 peut être amené à vouloir insérer de nouveaux n-uplets dans Stars que P_2 est en train de parcourir.
- P_2 ajoute de nouveaux n-uplets qui sont lus par P_1

Pour s'en prémunir, on écrit :

```
EXEC SQL DECLARE filmStarcurseur INSENSITIVE CURSOR FOR
```

Différents parcours au sein d'un curseur

1. A la déclaration

   `EXEC SQL DECLARE <curseur> SCROLL CURSOR FOR`

2. Différentes options pour le `FETCH`

   (a) `NEXT` ou `PRIOR`

   (b) `FIRST` ou `LAST`

   (c) `RELATIVE <+|-n>`

   (d) `ABSOLUTE`

Exemple

```
6) EXEC SQL DECLARE execCurseur SCROLL CURSOR FOR
7)      SELECT BeneficePres FROM Producteur;
   EXEC SQL FETCH LAST FROM execCurseur INTO :benef;
9) while(1) {
/* lignes  11) A 15) inchangees */
   EXEC SQL FETCH PRIOR FROM execCurseur INTO :benef;
```

SQL Dynamique

```
1) void lireRequete() {
2)      EXEC SQL BEGIN DECLARE SECTION;
3)           char *requete;
4)      EXEC SQL END DECLARE SECTION;
5) /* demande a l'utilisateur d'entrer le texte
      de sa requete. alloue de la memoire (malloc)
      et fait pointer :requete sur le 1er caractere
      de la requete */
6)      EXEC SQL PREPARE SQLquery FROM :requete;
7)      EXEC SQL EXECUTE SQLquery;
}
```